Reminding

- 11/24(Thu) Final Project Announcement (in class)
- 12/8(Thu) HW2 Deadline & HW3 Announcement

  Submit your research topic (for group presentation)

How to search important papers in NLP?

- Use keywords to search the topics you have interest in it
  - https://aclanthology.org

- I would suggest you select some papers from the following list:
  - https://aclanthology.org/events/acl-2022/

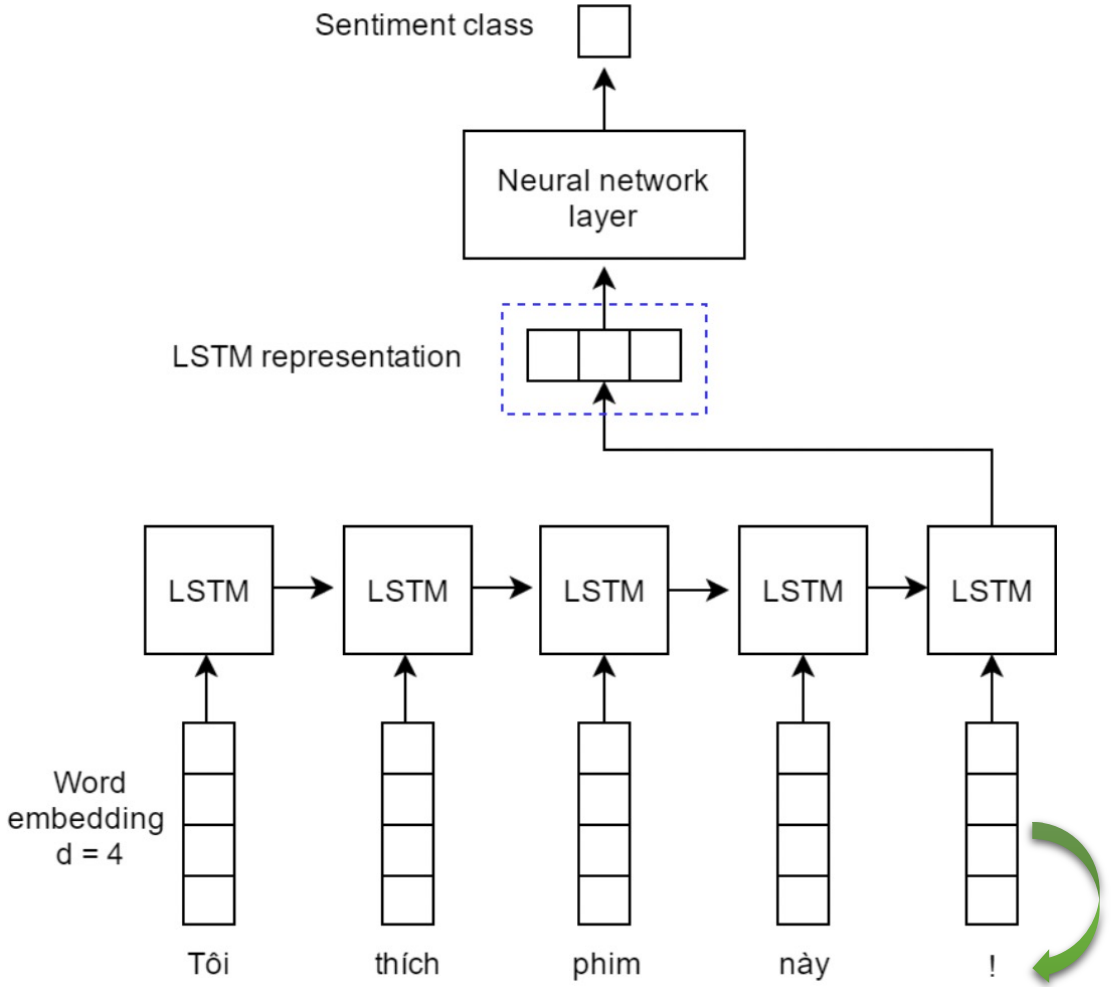
HW2

How to convert text to vectors? 🤔

➔ You can use word embeddings

➔ To download word embeddings, please refer to page 59

An example of framework for sentiment analysis



The vector of the corresponding word

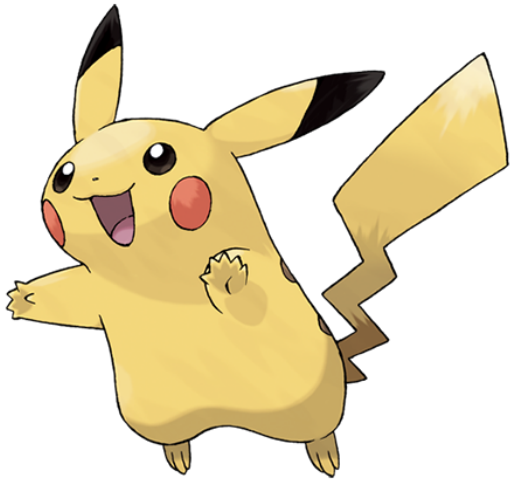
http://dx.doi.org/10.1109/KSE.2017.8119429

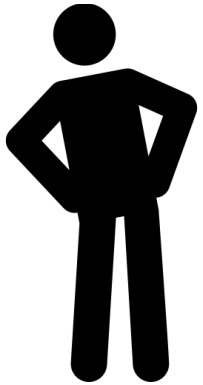
Word Embeddings

顏安孜

azyen@cs.nycu.edu.tw

Word Semantics





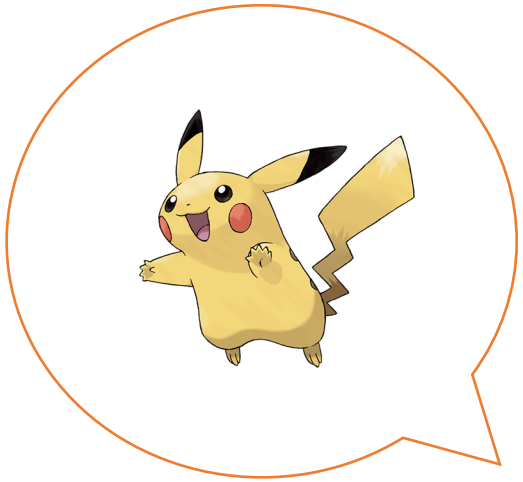
It is a Pikachu.
Pikachu that can generate powerful electricity have cheek sacs that are extra soft and super stretchy.

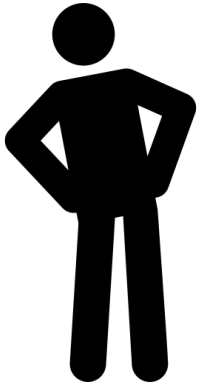
Words are just strings

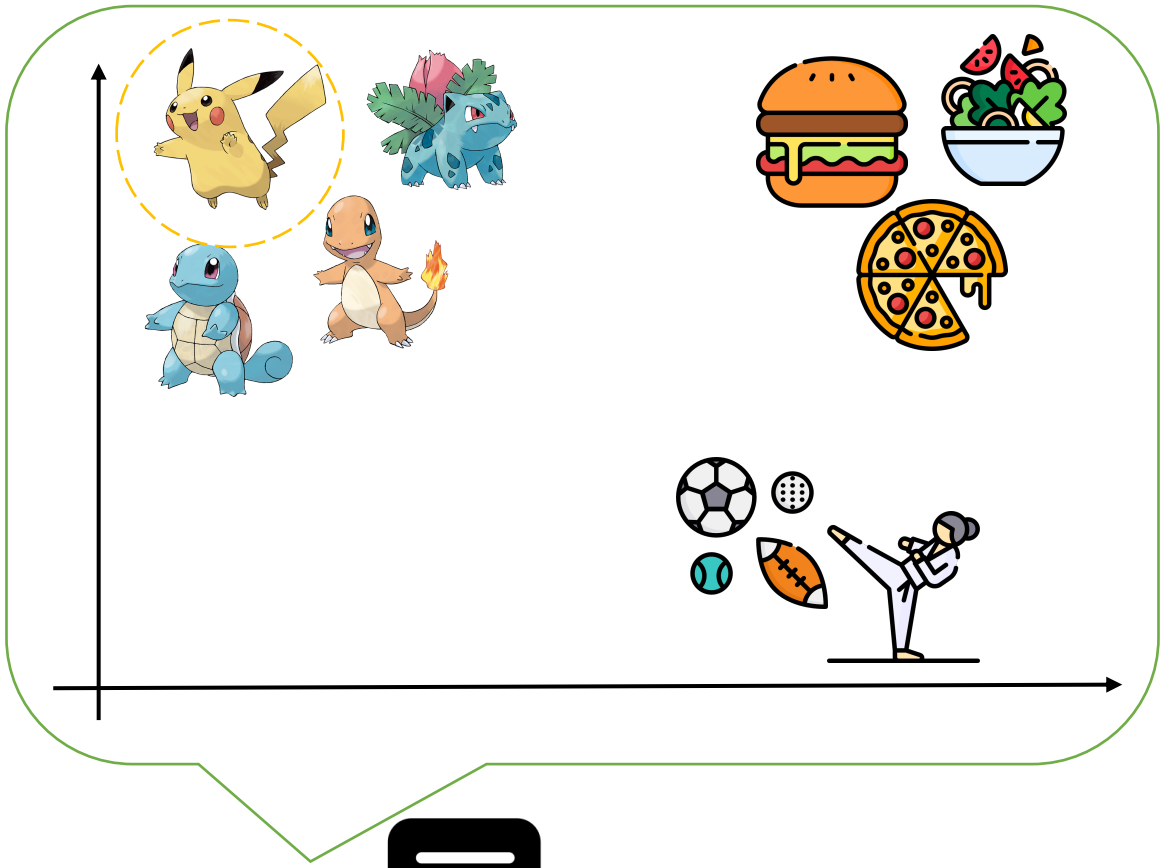
0000110101…

Source of image: https://www.pokemon.com/us/

Word Vectors







Source of image: https://www.pokemon.com/us/

Meaning Representations

- Where are you going?
- Where are you from?

} Different meaning

- What is your age?
- How old are you?

} Same meaning

Meaning Representations in Computers

- Knowledge-Based Representation
- Corpus-Based Representation

Knowledge-Based Representation–WordNet

- A large online thesaurus that represents word senses
  - e.g., IS-A relation (*dog* and *mammal*), part-whole relation (*engine* and *car*)

Benz is credited with the invention of the *motorcar*.

Benz is credited with the invention of the *automobile*.

➔ The meaning of the sentence stays pretty much the same

```
1 from nltk.corpus import wordnet as wn
```

```
1 wn.synsets('motorcar')
```

```
[Synset('car.n.01')]
```

motorcar has just one possible meaning (car.n.01)

```
1 wn.synset('car.n.01').lemma_names()
```

```
['car', 'auto', 'automobile', 'machine', 'motorcar']
```

```
1 wn.synset('car.n.01').definition()
```

```
'a motor vehicle with four wheels; usually propelled by an internal combustion engine'
```

```
1 wn.synset('car.n.01').examples()
```

```
['he needs a car to get to work']
```

To eliminate ambiguity, we will identify these words as *car.n.01.automobile*, *car.n.01.motorcar*

```
1 wn.synset('car.n.01').lemmas()
```

```
[Lemma('car.n.01.car'),
 Lemma('car.n.01.auto'),
 Lemma('car.n.01.automobile'),
 Lemma('car.n.01.machine'),
 Lemma('car.n.01.motorcar')]
```

```
1 wn.synsets('automobile')
```

```
[Synset('car.n.01'), Synset('automobile.v.01')]
```

```
1 # the word car is ambiguous
2 wn.synsets('car')
```

```
[Synset('car.n.01'),
 Synset('car.n.02'),
 Synset('car.n.03'),
 Synset('car.n.04'),
 Synset('cable_car.n.01')]
```

```
1 for synset in wn.synsets('car'):
2     print(synset.lemma_names())
```

```
['car', 'auto', 'automobile', 'machine', 'motorcar']
['car', 'railcar', 'railway_car', 'railroad_car']
['car', 'gondola']
['car', 'elevator_car']
['cable_car', 'car']
```

有軌機動車

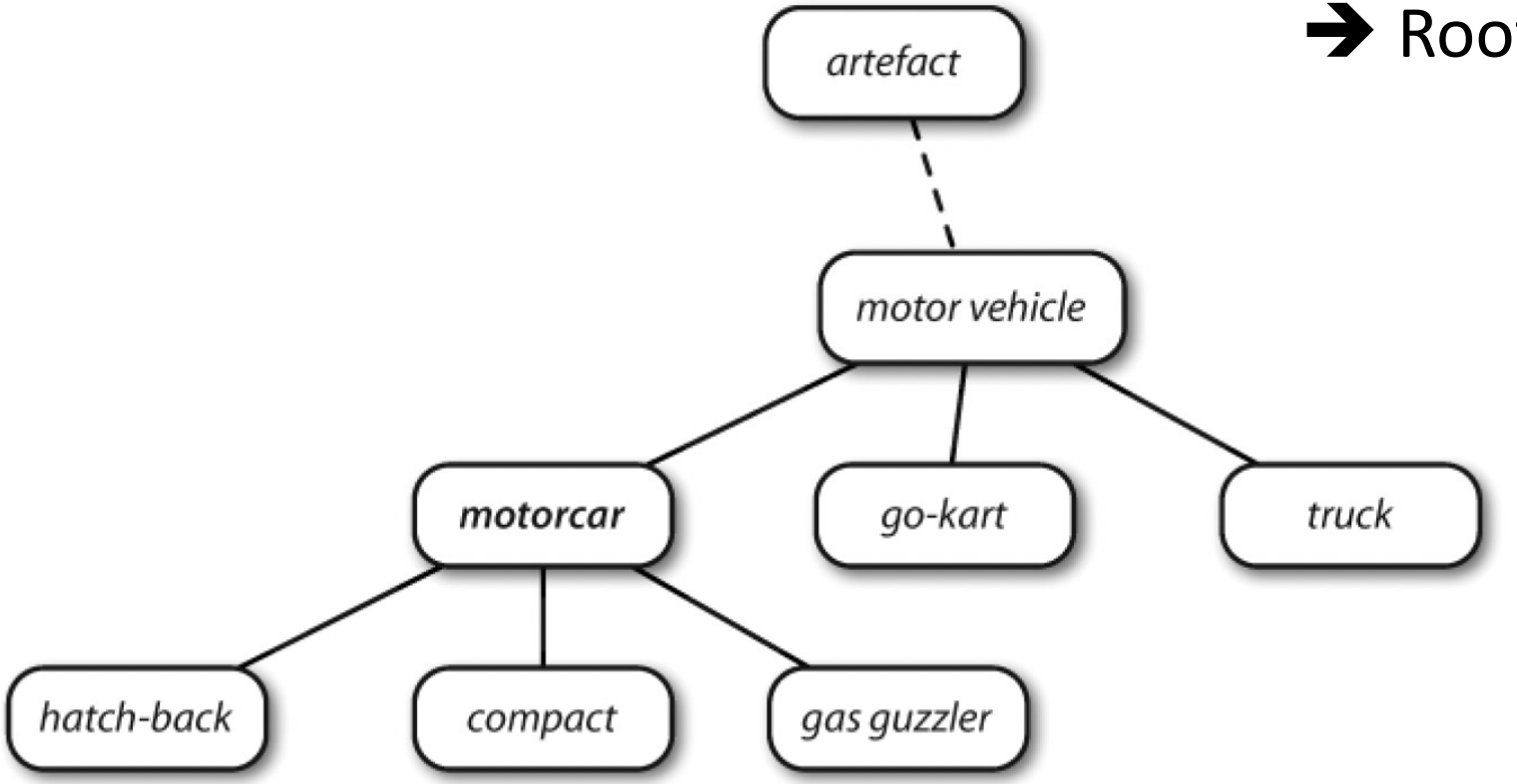
（用於義大利威尼斯運河的）平底狹長小船；（鐵路上的）敞篷貨車

轎廂電梯或車輛電梯是設計用於垂直運輸車輛的電梯

- The word *car* is ambiguous, having five synsets

The WordNet Hierarchy

General concepts:
Entity, State, Event
➔ Root synsets



- Nodes correspond to synsets; edges indicate the hypernym/hyponym relation

Hyponyms

```
1 motorcar = wn.synset('car.n.01')
```

```
1 types_of_motorcar = motorcar.hyponyms()
```

```
1 types_of_motorcar
```

```
[Synset('ambulance.n.01'),
 Synset('beach_wagon.n.01'),
 Synset('bus.n.04'),
 Synset('cab.n.03'),
 Synset('compact.n.03'),
 Synset('convertible.n.01'),
 Synset('coupe.n.01'),
 Synset('cruiser.n.01'),
 Synset('electric.n.01'),
 Synset('gas_guzzler.n.01'),
 Synset('hardtop.n.01'),
 Synset('hatchback.n.01'),
 Synset('horseless_carriage.n.01'),
 Synset('hot_rod.n.01'),
 Synset('jeep.n.01'),
 Synset('limousine.n.01'),
 Synset('loaner.n.02'),
```

Hypernyms

```
1 motorcar.hypernyms()
```

```
[Synset('motor_vehicle.n.01')]
```

```
1 paths = motorcar.hypernym_paths()
2 len(paths)
```

```
2
```

```
1 [synset.name() for synset in paths[0]]
```

```
['entity.n.01',
 'physical_entity.n.01',
 'object.n.01',
 'whole.n.02',
 'artifact.n.01',
 'instrumentality.n.03',
 'container.n.01',
 'wheeled_vehicle.n.01',
 'self-propelled_vehicle.n.01',
 'motor_vehicle.n.01',
 'car.n.01']
```

```
1 wheeled_vehicle = wn.synset('wheeled_vehicle.n.01')
```

```
1 wheeled_vehicle.hypernyms()
```

```
[Synset('container.n.01'), Synset('vehicle.n.01')]
```

```
1 [synset.name() for synset in paths[1]]
```

```
['entity.n.01',
 'physical_entity.n.01',
 'object.n.01',
 'whole.n.02',
 'artifact.n.01',
 'instrumentality.n.03',
 'conveyance.n.03',
 'vehicle.n.01',
 'wheeled_vehicle.n.01',
 'self-propelled_vehicle.n.01',
 'motor_vehicle.n.01',
 'car.n.01']
```

```
1 motorcar.root_hypernyms()
```

```
[Synset('entity.n.01')]
```

Semantic Similarity

```
1  milk = wn.synset('milk.n.01')
2  dog = wn.synset('dog.n.01')
3  cat = wn.synset('cat.n.01')
4  bird = wn.synset('bird.n.01')
5  spider = wn.synset('spider.n.01')
6  fish = wn.synset('fish.n.01')
7  pig = wn.synset('pig.n.01')
8  ship = wn.synset('ship.n.01')
9  bee = wn.synset('bee.n.01')
10 ant = wn.synset('ant.n.01')
11 egg = wn.synset('egg.n.01')
12 duck = wn.synset('duck.n.01')
```

```
1 print(cat.lowest_common_hypernyms(dog))
```

```
[Synset('carnivore.n.01')]
```
肉食動物

```
1 print(cat.lowest_common_hypernyms(bird))
```

```
[Synset('vertebrate.n.01')]
```
脊椎動物

```
1 print(bee.lowest_common_hypernyms(ant))
```

```
[Synset('hymenopterous_insect.n.01')]
```
昆蟲

```
1 print(bee.lowest_common_hypernyms(spider))
```

```
[Synset('arthropod.n.01')]
```
節肢動物

```
1 print(bird.lowest_common_hypernyms(duck))
```

```
[Synset('bird.n.01')]
```
鳥類

```
1 print(bird.lowest_common_hypernyms(egg))
```

```
[Synset('living_thing.n.01')]
```
生物

```
1 novel = wn.synset('novel.n.01')
2 print(bird.lowest_common_hypernyms(novel))
```

```
[Synset('entity.n.01')]
```

The depth of each synset

```
1 wn.synset('carnivore.n.01').min_depth()
```

```
11
```

肉食動物

```
1 wn.synset('vertebrate.n.01').min_depth()
```

```
8
```

脊椎動物

```
1 wn.synset('hymenopterous_insect.n.01').min_depth()
```

```
10
```

昆蟲

```
1 wn.synset('arthropod.n.01').min_depth()
```

```
8
```

節肢動物

```
1 wn.synset('bird.n.01').min_depth()
```

```
9
```

鳥類

```
1 wn.synset('living_thing.n.01').min_depth()
```

```
4
```

生物

```
1 wn.synset('entity.n.01').min_depth()
```

```
0
```

Path Similarity

- path_similarity assigns a score in the range 0–1 based on the shortest path that connects the concepts in the hypernym hierarchy

```
1 print(cat.path_similarity(dog))
```

```
0.2
```

```
1 print(cat.path_similarity(bird))
```

```
0.1428571428571428
```

```
1 print(cat.path_similarity(ant))
```

```
0.07692307692307693
```

```
1 print(cat.path_similarity(egg))
```

```
0.06666666666666667
```

```
1 print(cat.path_similarity(novel))
```

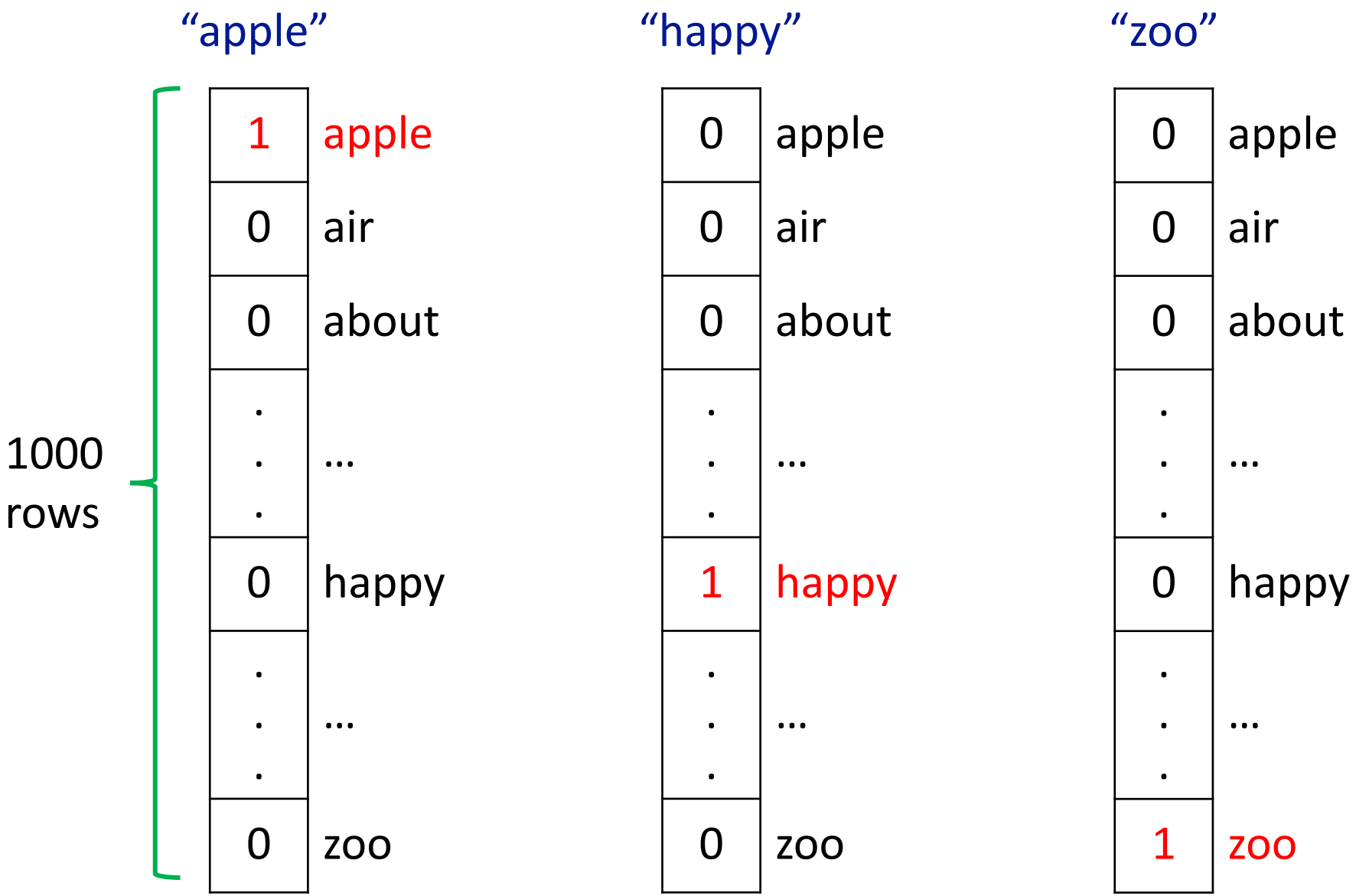
```
0.047619047619047616
```

Knowledge-Based Representation

- Issues:
  - newly-invented words
  - subjective
  - annotation effort
  - difficult to compute word similarity

Corpus-Based Representation

one-hot representation



TF-IDF

- TF-IDF is an information retrieval method that relies on Term Frequency (TF) and Inverse Document Frequency (IDF) to measure the importance of a word in a document.
- A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use TF-IDF.

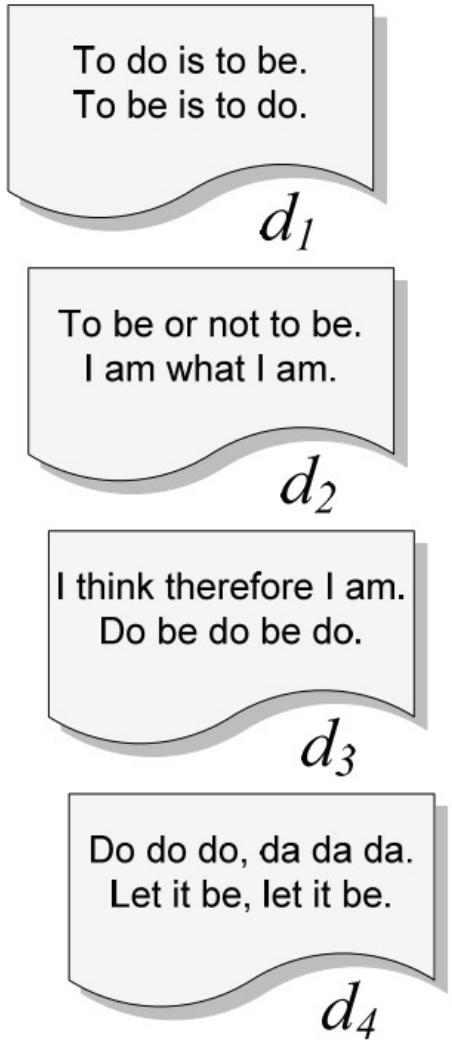
Term Frequency (TF) Weights

- A variant of tf weight used in the literature is

$$tf_{i,j} = \begin{cases} 1 + \log f_{i,j} & \texttt{if} \quad f_{i,j} > 0 \\ 0 & \texttt{otherwise} \end{cases}$$

where the log is taken in base 2

Term Frequency (TF) Weights



Vocabulary	
1	to
2	do
3	is
4	be
5	or
6	not
7	I
8	am
9	what
10	think
11	therefore
12	da
13	let
14	it

$tf_{i,1}$	$tf_{i,2}$	$tf_{i,3}$	$tf_{i,4}$
3	2	-	-
2	-	2.585	2.585
2	-	-	-
2	2	2	2
-	1	-	-
-	1	-	-
-	2	2	-
-	2	1	-
-	1	-	-
-	-	1	-
-	-	1	-
-	-	-	2.585
-	-	-	2
-	-	-	2

1 + log 4

1 + log 3

=1 + 1.5849

log is taken in base 2

Inverse Document Frequency

- The more index terms are assigned to a document, the higher is the probability of retrieval for that document
- If too many terms are assigned to a document, it will be retrieved by queries for which it is not relevant
- Let k_i be the term with the rth largest document frequency, i.e., $n(r) = n_i$. Then,

$$idf_i = \log \frac{N}{n_i}$$

where idf_i is called the inverse document frequency of term k_i

Words like "the" or "it" have very low idf

TF-IDF weighting scheme

- Let $w_{i,j}$ be the term weight associated with the term k_i and the document d_j
- Then, we define

TF

$$w_{i,j} = \begin{cases} (1 + \log f_{i,j}) \times \log \frac{N}{n_i} & \texttt{if } f_{i,j} > 0 \\ 0 & \texttt{otherwise} \end{cases}$$

which is referred to as a tf-idf weighting scheme

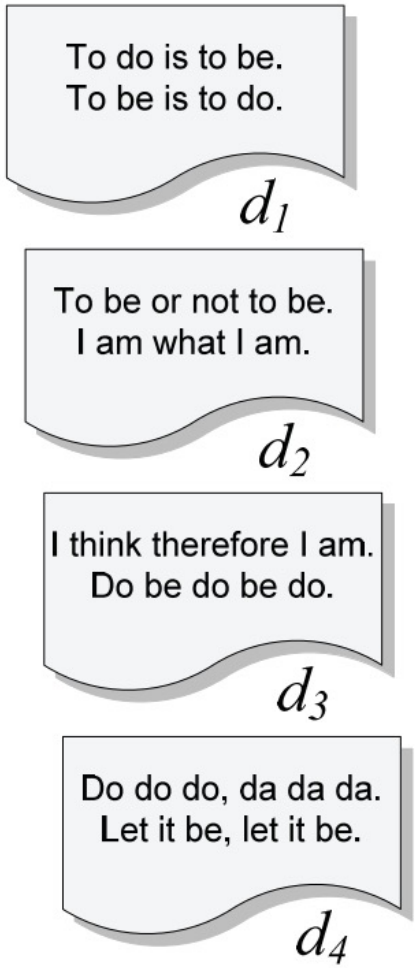
TF-IDF weighting scheme

- Tf-idf weights of all terms present in our example document collection

Vocabulary	
1	to
2	do
3	is
4	be
5	or
6	not
7	I
8	am
9	what
10	think
11	therefore
12	da
13	let
14	it

$tf_{i,1}$	$tf_{i,2}$	$tf_{i,3}$	$tf_{i,4}$
3	2	-	-
2	-	2.585	2.585
2	-	-	-
2	2	2	2
-	1	-	-
-	1	-	-
-	2	2	-
-	2	1	-
-	1	-	-
-	-	1	-
-	-	1	-
-	-	-	2.585
-	-	-	2
-	-	-	2

	term	n_i	$idf_i = \log(N/n_i)$
1	to	2	1
2	do	3	0.415
3	is	1	2
4	be	4	0
5	or	1	2
6	not	1	2
7	I	2	1
8	am	2	1
9	what	1	2
10	think	1	2
11	therefore	1	2
12	da	1	2
13	let	1	2
14	it	1	2



3 × 1

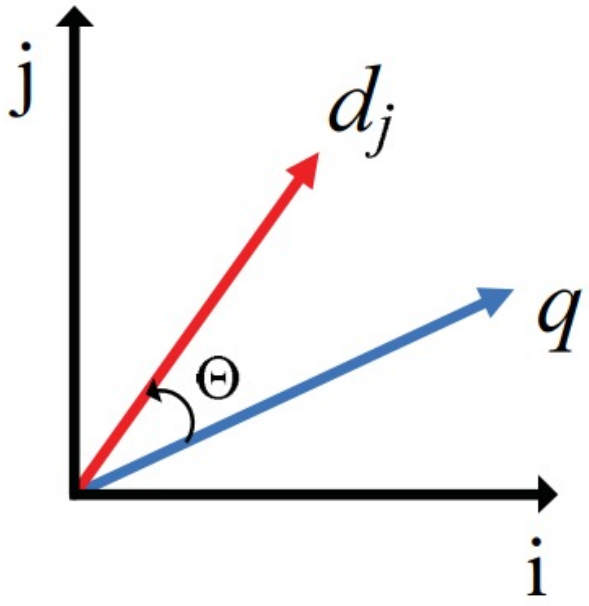
		d_1	d_2	d_3	d_4
1	to	3	2	-	-
2	do	0.830	-	1.073	1.073
3	is	4	-	-	-
4	be	-	-	-	-
5	or	-	2	-	-
6	not	-	2	-	-
7	I	-	2	2	-
8	am	-	2	1	-
9	what	-	2	-	-
10	think	-	-	2	-
11	therefore	-	-	2	-
12	da	-	-	-	5.170
13	let	-	-	-	4
14	it	-	-	-	4

2 × 0.415

2 × 0

The Vector Model

- Similarity between a document d_j and a query q



The document length is given by the norm of this vector, which is computed as follows

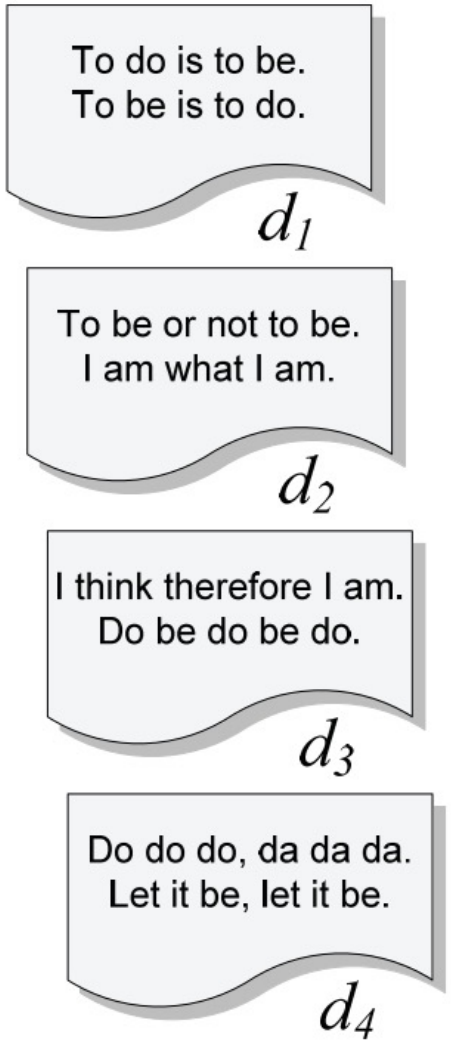
$$|\vec{d_j}| = \sqrt{\sum_{i}^{t} w_{i,j}^2}$$

$$cos(\theta) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|}$$

$$sim(d_j, q) = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}$$

$$0 \leqslant sim(d_j, q) \leqslant 1$$

Document Length Normalization



TF-IDF:

		d_1	d_2	d_3	d_4
1	to	3	2	-	-
2	do	0.830	-	1.073	1.073
3	is	4	-	-	-
4	be	-	-	-	-
5	or	-	2	-	-
6	not	-	2	-	-
7	I	-	2	2	-
8	am	-	2	1	-
9	what	-	2	-	-
10	think	-	-	2	-
11	therefore	-	-	2	-
12	da	-	-	-	5.170
13	let	-	-	-	4
14	it	-	-	-	4

	d_1	d_2	d_3	d_4
size in bytes	34	37	41	43
number of words	10	11	10	12
vector norm	5.068	4.899	3.762	7.738

$(3^2+0.830^2+4^2)^{1/2}$

$(1.073^2+2^2+1^2+2^2+2^2)^{1/2}$

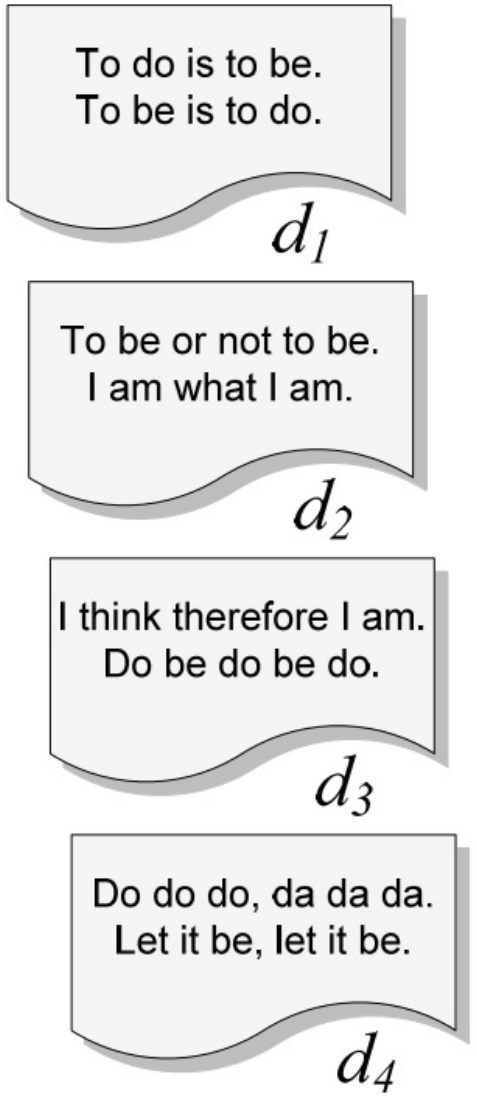
The Vector Model

- Document ranks computed by the Vector model for the query “to do”

Vocabulary	
1	to
2	do
3	is
4	be
5	or
6	not
7	I
8	am
9	what
10	think
11	therefore
12	da
13	let
14	it

$tf_{i,1}$	$tf_{i,2}$	$tf_{i,3}$	$tf_{i,4}$
3	2	-	-
2	-	2.585	2.585
2	-	-	-
2	2	2	2
-	1	-	-
-	1	-	-
-	2	2	-
-	2	1	-
-	1	-	-
-	-	1	-
-	-	1	-
-	-	-	2.585
-	-	-	2
-	-	-	2

	term	n_i	$idf_i = \log(N/n_i)$
1	to	2	1
2	do	3	0.415
3	is	1	2
4	be	4	0
5	or	1	2
6	not	1	2
7	I	2	1
8	am	2	1
9	what	1	2
10	think	1	2
11	therefore	1	2
12	da	1	2
13	let	1	2
14	it	1	2

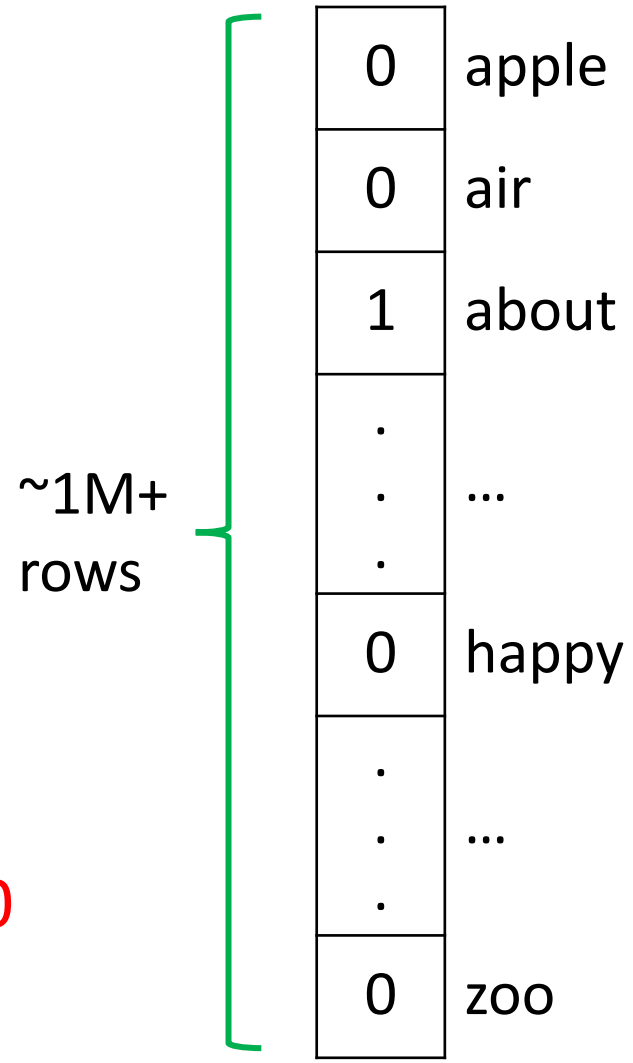


doc	rank computation	rank
d_1	$\frac{1*3+0.415*0.830}{5.068}$	0.660
d_2	$\frac{1*2+0.415*0}{4.899}$	0.408
d_3	$\frac{1*0+0.415*1.073}{3.762}$	0.118
d_4	$\frac{1*0+0.415*1.073}{7.738}$	0.058

	d_1	d_2	d_3	d_4
size in bytes	34	37	41	43
number of words	10	11	10	12
vector norm	5.068	4.899	3.762	7.738

One-hot Vectors

- Simple
- No implied ordering
- Huge vectors
- No semantic meaning



[0 0 0 0 1 0 0 … 0 0] AND [0 0 0 0 0 1 0 … 0 0] = 0

cat lion

Issues: difficult to compute the similarity (i.e. comparing “car” and “motorcycle”)

Corpus-Based Representation

- Idea: words with similar meanings often have similar neighbors
- Neighbor-based representation
  - Co-occurrence matrix constructed via neighbors
  - Neighbor definition: full document v.s. windows

  - full document
    - word-document co-occurrence matrix gives general topics
    - Latent Semantic Analysis
  - windows
    - context window for each word
    - capture syntactic (e.g. POS) and semantic information

Window-Based Co-occurrence Matrix

- Example
  - Window length=1
  - Left or right context
  - Corpus:
    - I love AI.
    - I love deep learning.
    - I enjoy learning.

similarity > 0

Counts	I	love	enjoy	AI	deep	learning
I	0	2	1	0	0	0
love	2	0	0	1	1	0
enjoy	1	0	0	0	0	1
AI	0	1	0	0	0	0
deep	0	1	0	0	0	1
learning	0	0	1	0	1	0

But most elements are zero 🥺

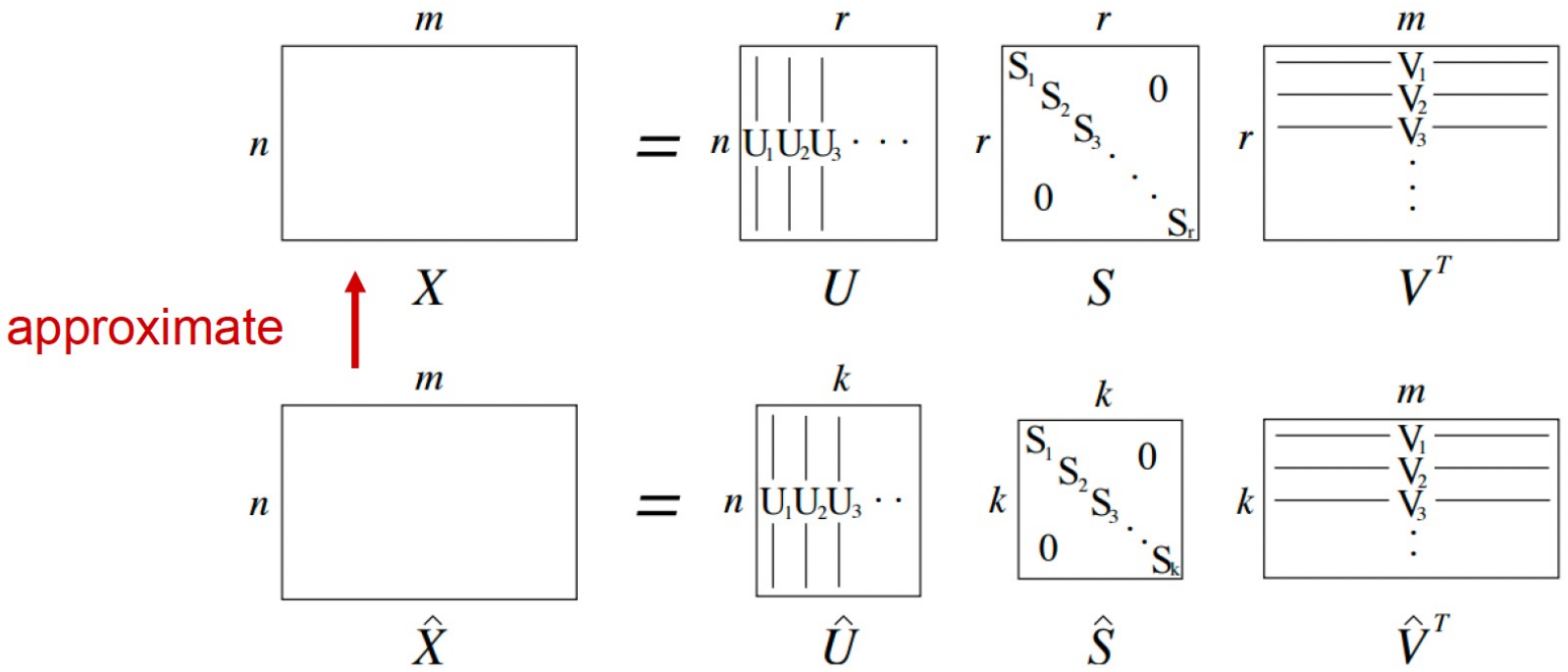
Issues:

- matrix size increases with vocabulary
- high dimensional
- sparsity ➔ poor robustness

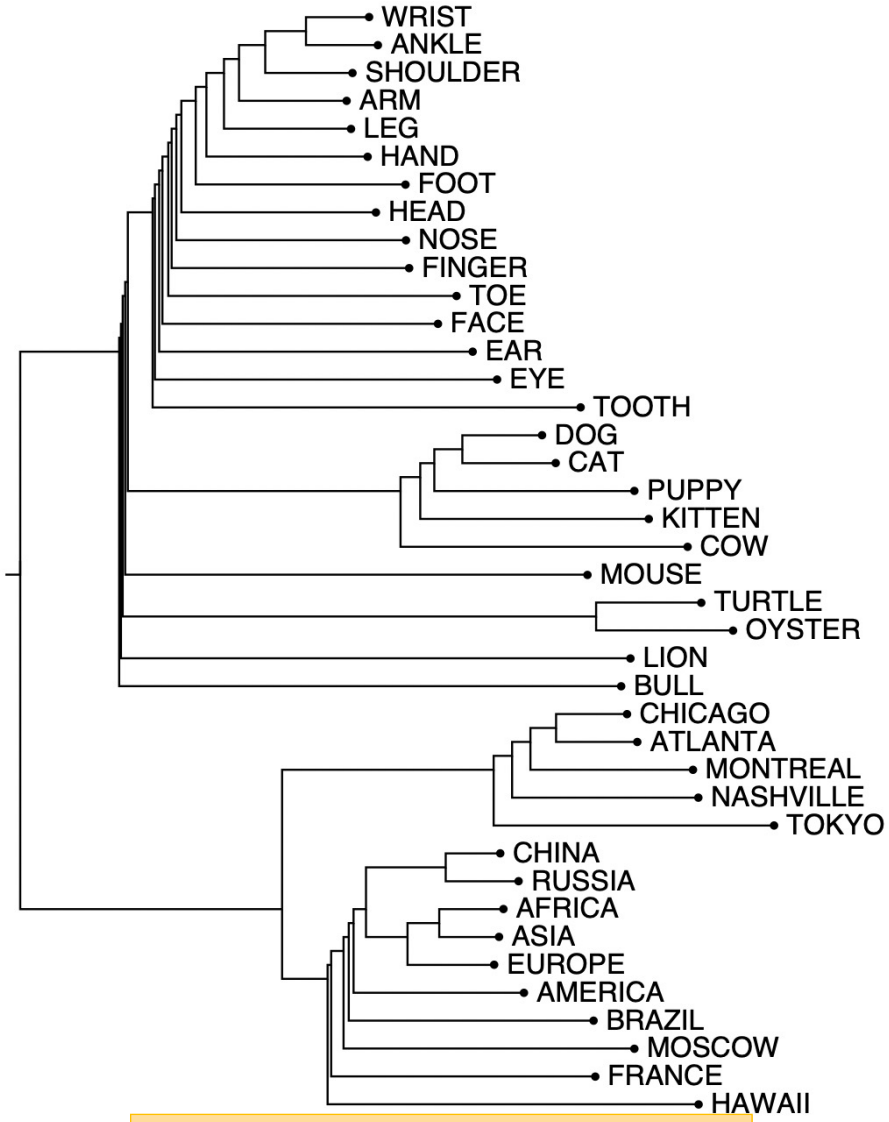
➔ Idea: low dimensional word vector

Low-Dimensional Dense Word Vector

- Method 1: dimension reduction on the matrix
- ➔ Singular Value Decomposition (SVD)



Low-Dimensional Dense Word Vector



Semantic Relations

Syntactic Relations

Rohde et al., "An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence," 2005.

Issues:

- computationally expensive: $O(mn^2)$ when $n<m$ for nxm matrix
- difficult to add new words

Low-Dimensional Dense Word Vector

- Method 2: directly learn low-dimensional word vectors
  - Learning representations by back-propagation. (Rumelhart et al., 1986)
  - A neural probabilistic language model (Bengio et al., 2003)
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
  - Recent and most popular models: word2vec (Mikolov et al. 2013) and Glove (Pennington et al., 2014)
    - As known as “Word Embeddings”

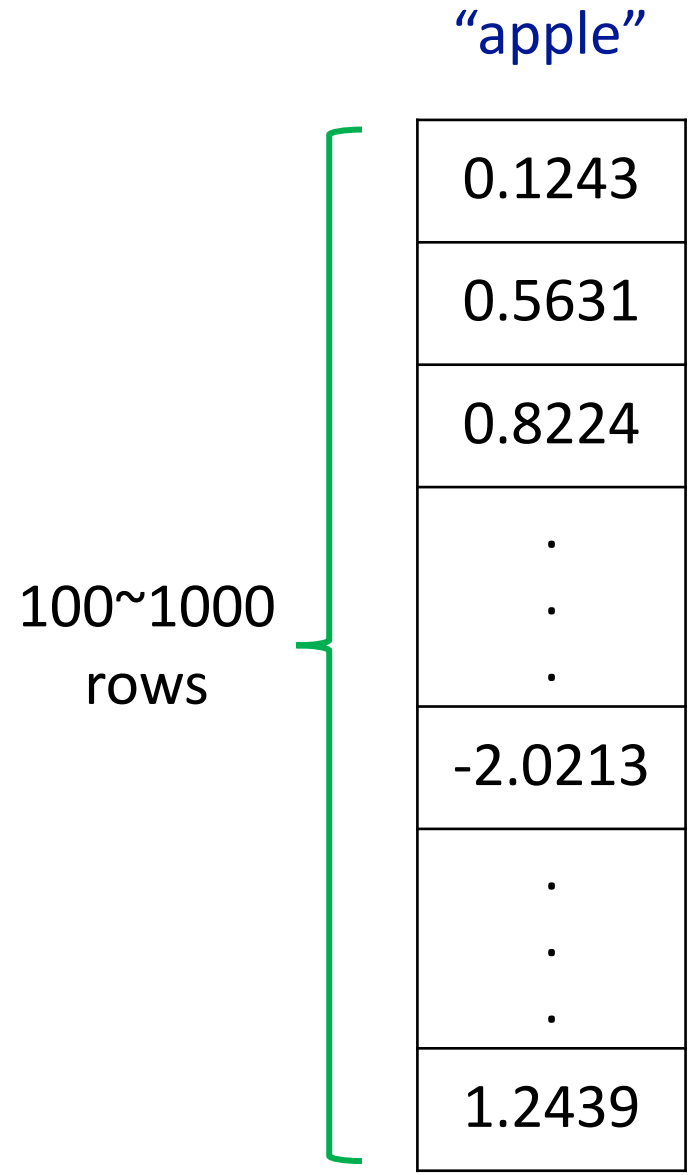
Word Embedding

- Machine learns the meaning of words from reading a lot of documents without supervision
- A word can be understood by its context

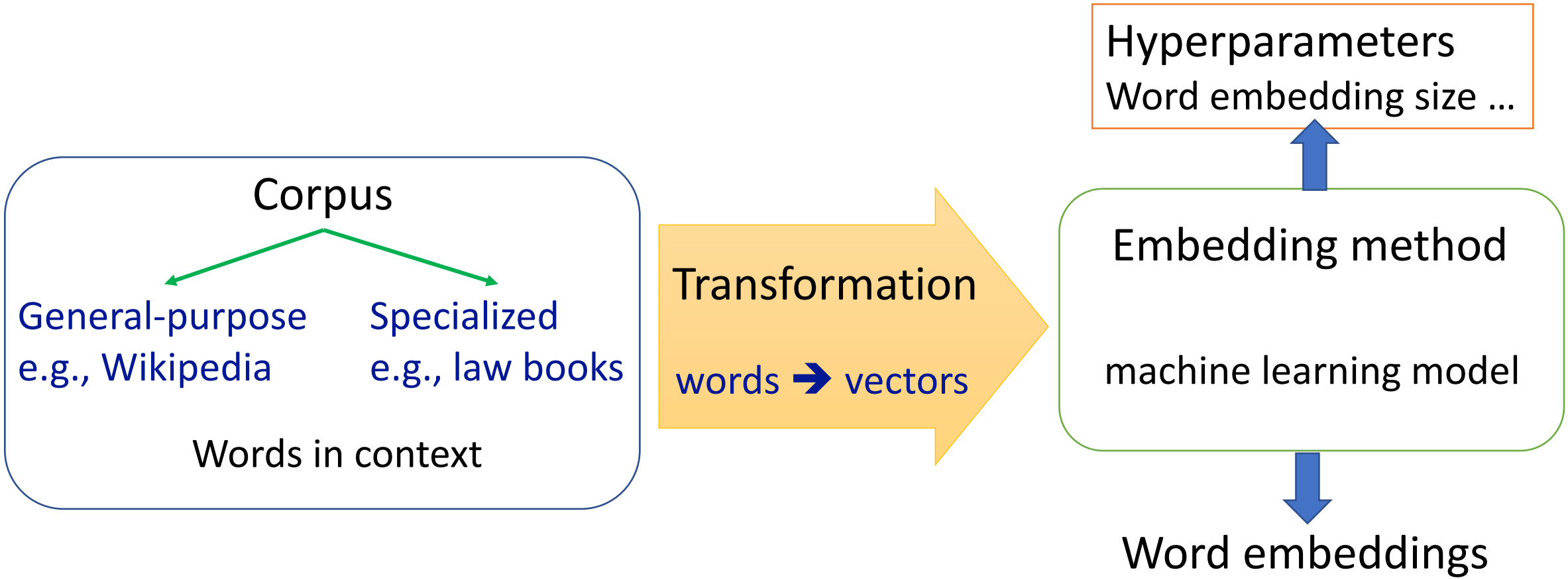
- 馬英九 520宣誓就職
- 蔡英文 520宣誓就職
- ➔ 蔡英文、馬英九 are something very similar

Word Embedding Vectors

- Low dimension
- Embed meaning
  - Semantic distance
    - desk ≈ table desk ≠ happy
  - Analogies
    - Japan : Tokyo = Taiwan : ? A: Taipei



Word Embedding Process



Word2Vec – Skip-Gram Model

- Goal: predict surrounding words within a window of each word
- Objective function: maximize the probability of any context word given the current center word

$$w_1, w_2, \cdots, \underbrace{w_{t-m}, \cdots, w_{t-1}}, \underbrace{w_t}, \underbrace{w_{t+1}, \cdots, w_{t+m}}, \cdots, w_{T-1}, w_T$$

context window

w_I w_O

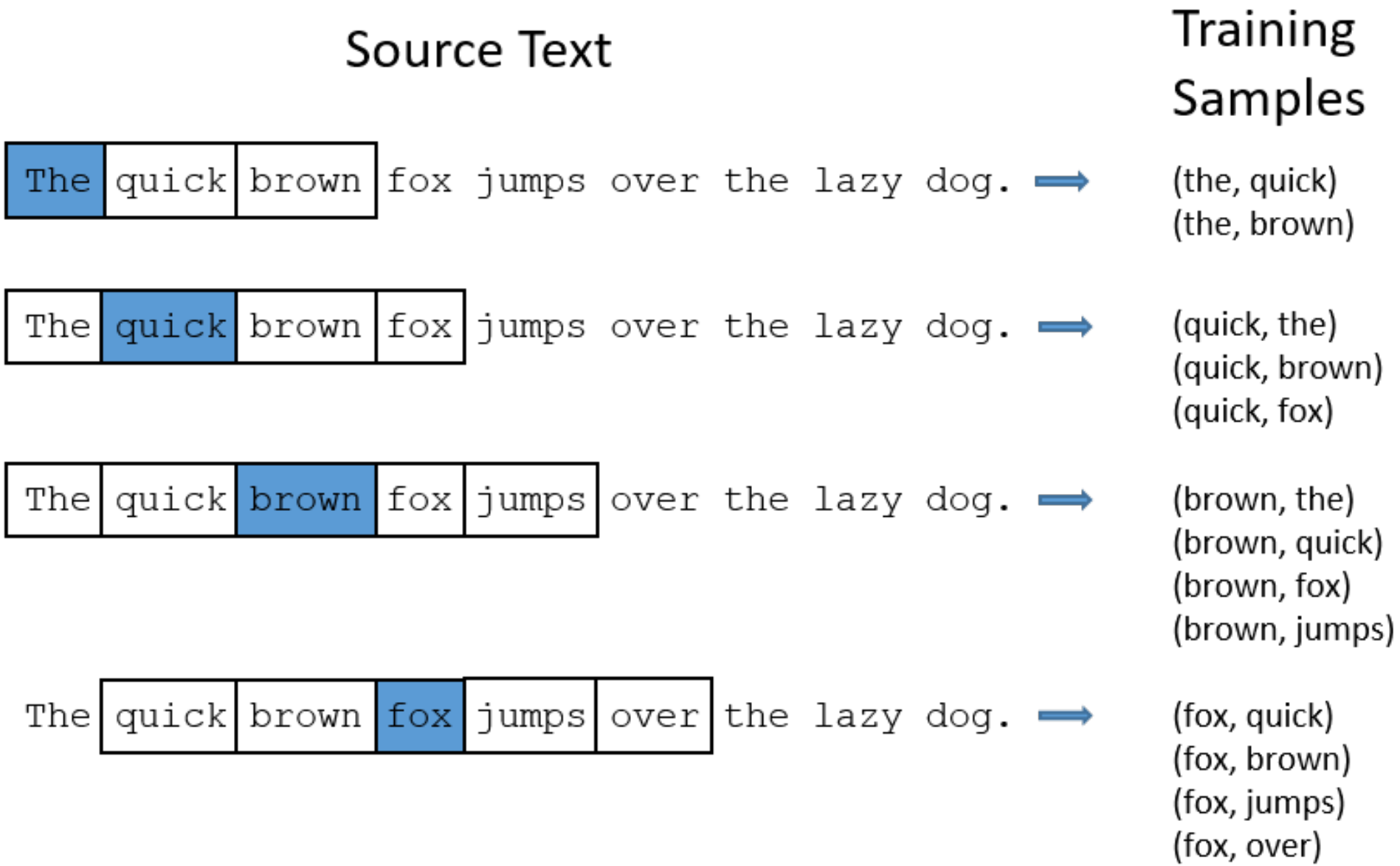
$$p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I) = \prod_{c=1}^{C} p(w_{O,c} \mid w_I)$$

$$C(\theta) = -\sum_{w_I} \sum_{c=1}^{C} \log p(w_{O,c} \mid w_I)$$

target word vector

$$p(w_O \mid w_I) = \frac{\exp(v'^T_{w_O} v_{w_I})}{\sum_j \exp(v'^T_{w_j} v_{w_I})}$$

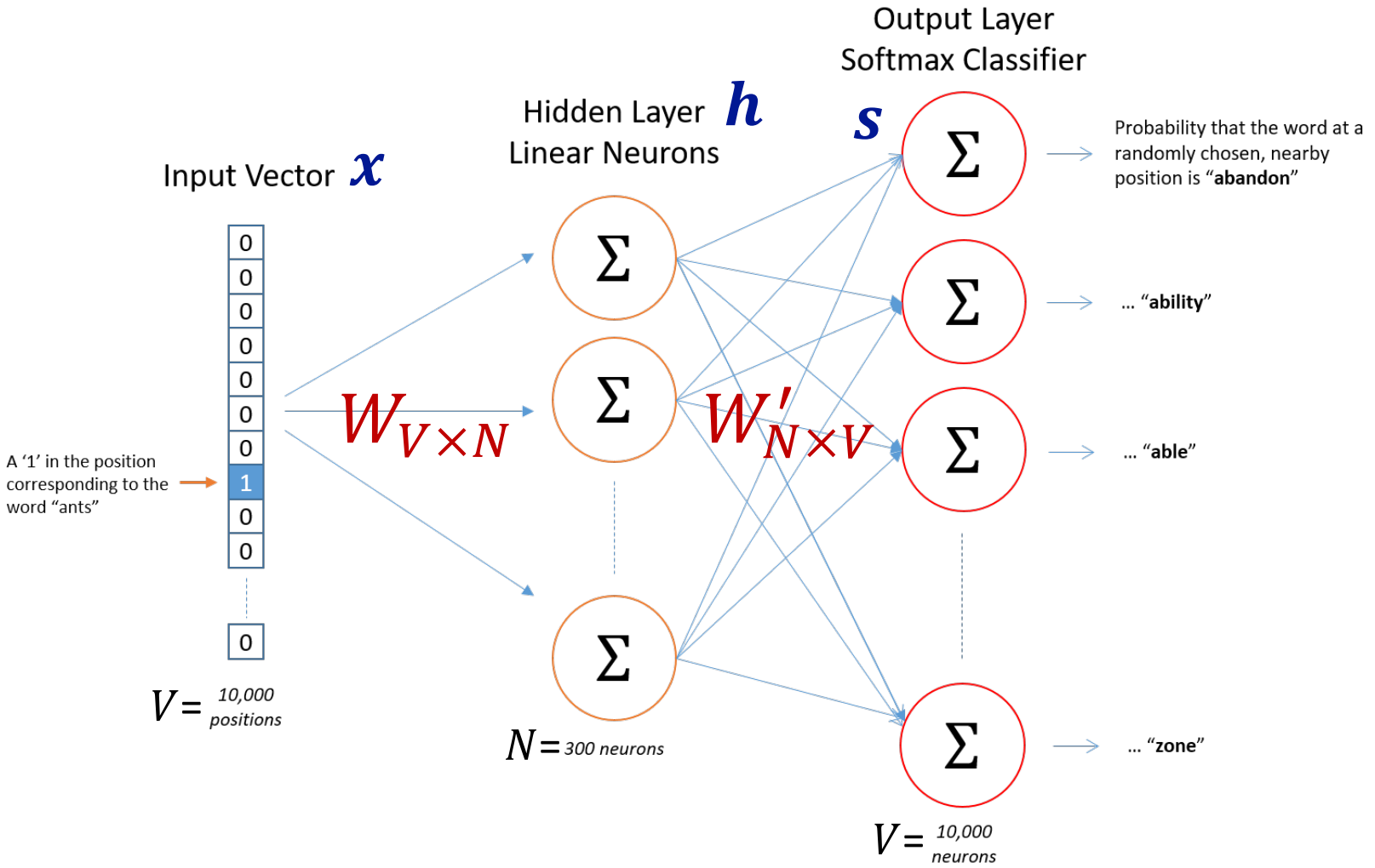
outside target word



Source of image: http://mccormickml.com/assets/word2vec/training_data.png

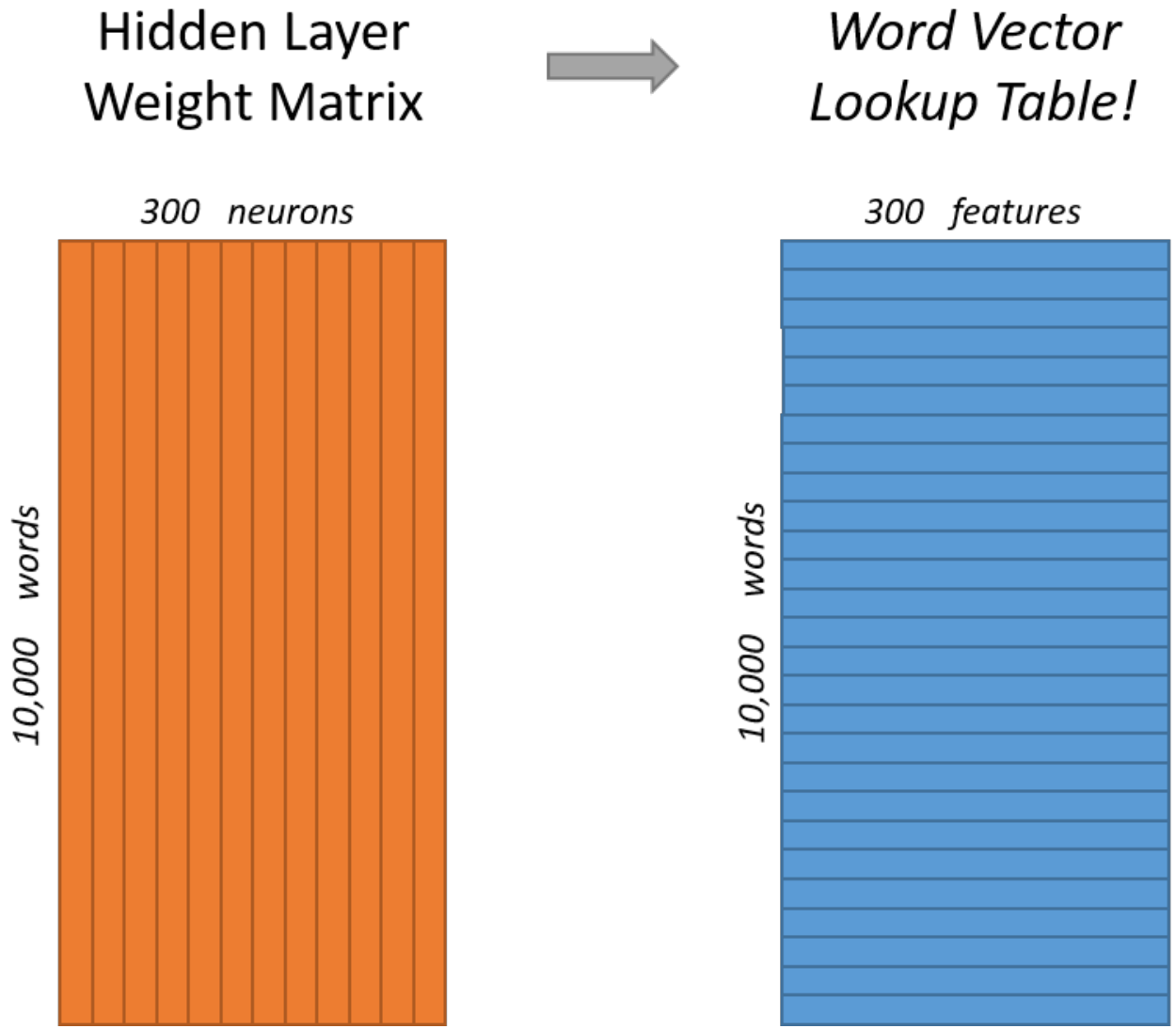
Word2Vec–Skip-Gram

- Goal: predict surrounding words within a window of each word



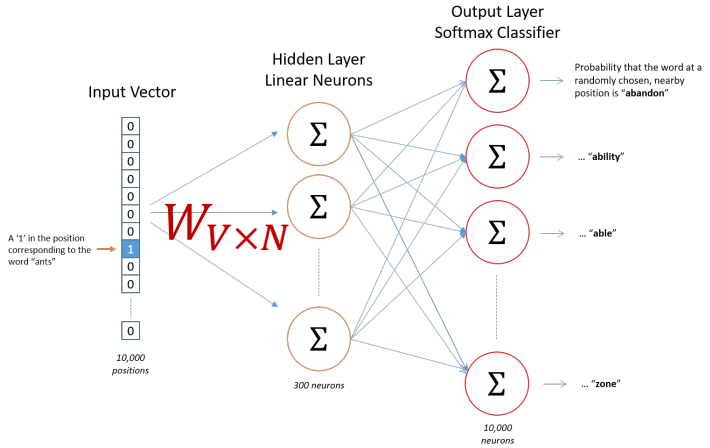
Source of image: http://mccormickml.com/assets/word2vec/skip_gram_net_arch.png

Hidden Layer Matrix → Word Embedding Matrix



Source of image: http://mccormickml.com/assets/word2vec/word2vec_weight_matrix_lookup_table.png

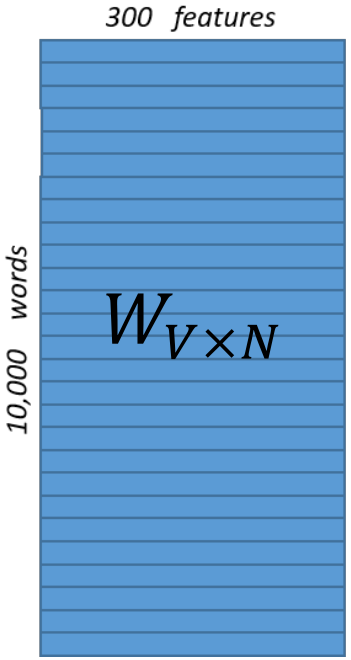
Weight Matrix Relation



- Hidden layer weight matrix = word vector lookup
- Each vocabulary entry has two vectors: as a target word and as a context word

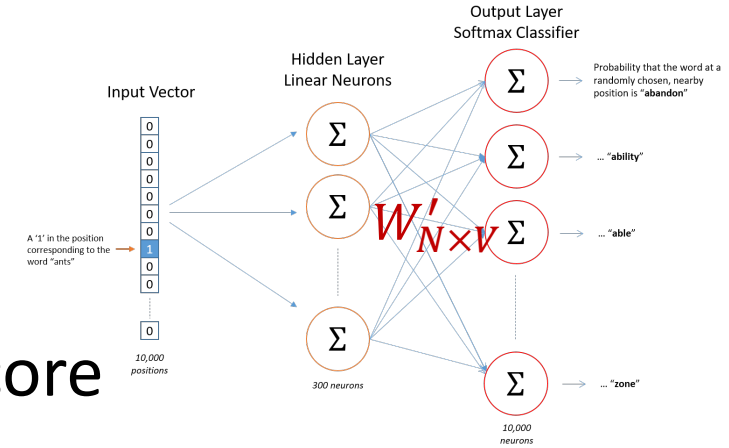
$$h = x^T W = W_{(k,.)} := v_{w_I}$$

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$



Source of image: http://mccormickml.com/assets/word2vec/matrix_mult_w_one_hot.png

Weight Matrix Relation



- Output layer weight matrix = weighted sum as final score

$$s_j = hv'_{w_j}$$

$$p(w_j = w_{O,c} \mid w_I) = y_{j_c} = \frac{\exp(s_{j_c})}{\sum_{j'=1}^{V} \exp(s_{j'})}$$

softmax

within the context window

Output weights for "car"

Word vector for "ants"

300 features

×

300 features

softmax

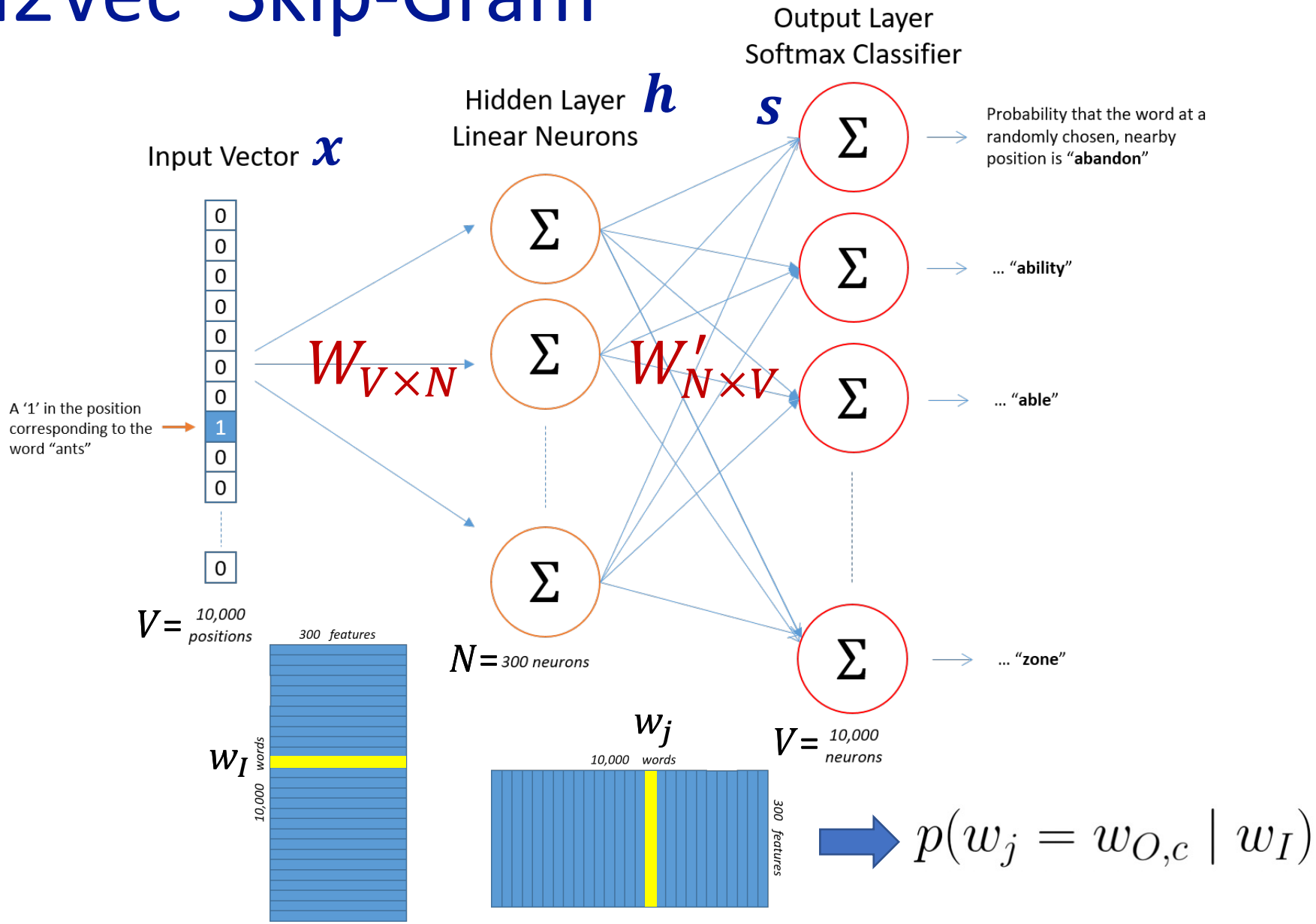
$\frac{e^x}{\sum e^x}$

=

Probability that if you randomly pick a word nearby "ants", that it is "car"

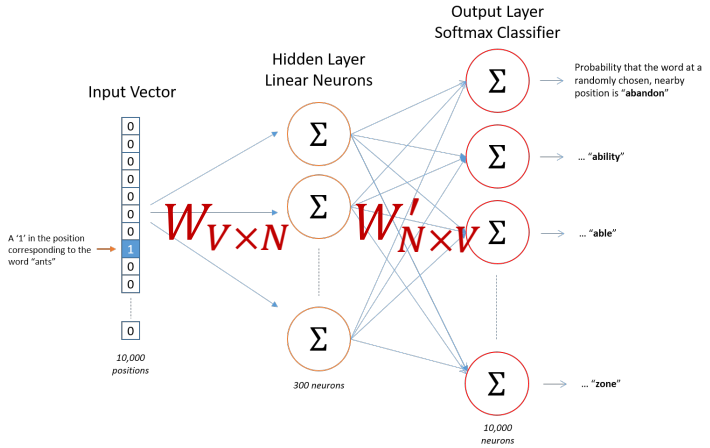
Source of image: http://mccormickml.com/assets/word2vec/output_weights_function.png

Word2Vec–Skip-Gram



Source of image: http://mccormickml.com/assets/word2vec/skip_gram_net_arch.png

Word2Vec Training–Loss Function



- Given a target word (w_I)

$$
\begin{aligned}
C(\theta) &= -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I) \\
&= -\log \prod_{c=1}^{C} \frac{\exp(s_{j_c})}{\sum_{j'=1}^{V} \exp(s_{j'})} \\
&= -\sum_{c=1}^{C} s_{j_c} + C \log \sum_{j'=1}^{V} \exp(s_{j'})
\end{aligned}
$$

Negative Sampling

- only update a sample of output vectors

$$C(\theta) = -\log \sigma({v'_{w_O}}^T v_{w_I}) + \sum_{w_j \in \mathcal{W}_{\text{neg}}} \log \sigma({v'_{w_j}}^T v_{w_I})$$

$${v'_{w_j}}^{(t+1)} = {v'_{w_j}}^{(t)} - \eta \cdot EI_j \cdot h$$

$$EI_j = \sigma({v'_{w_j}}^T v_{w_I}) - t_j$$

$$v_{w_I}^{(t+1)} = v_{w_I}^{(t)} - \eta \cdot EH^T$$

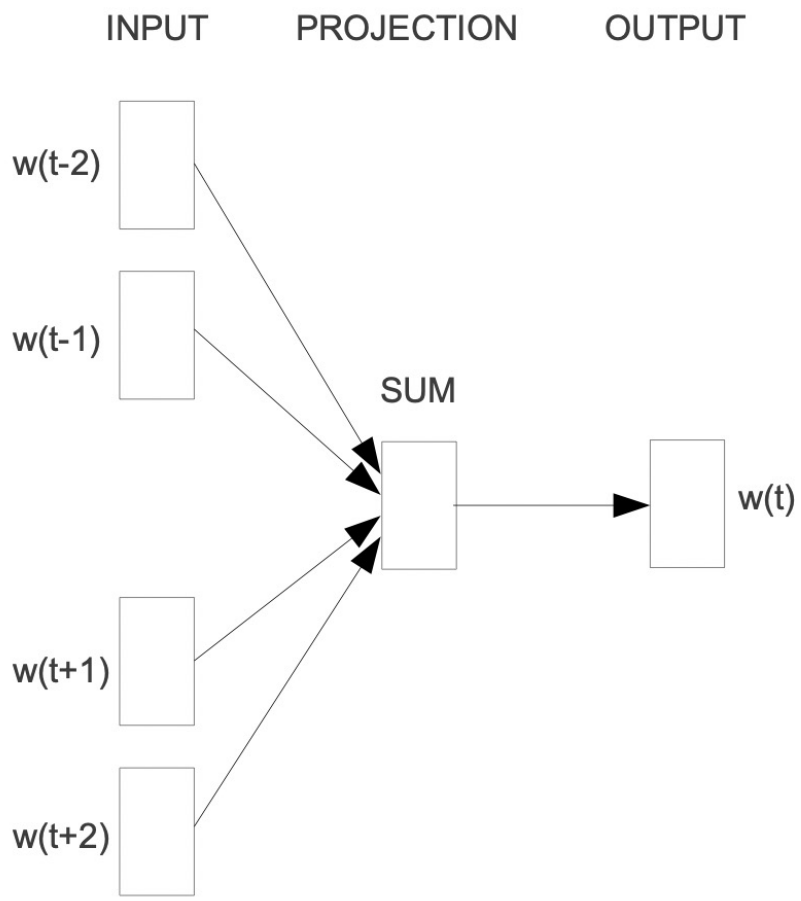
$$EH = \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} EI_j \cdot v'_{w_j}$$

$$w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}$$

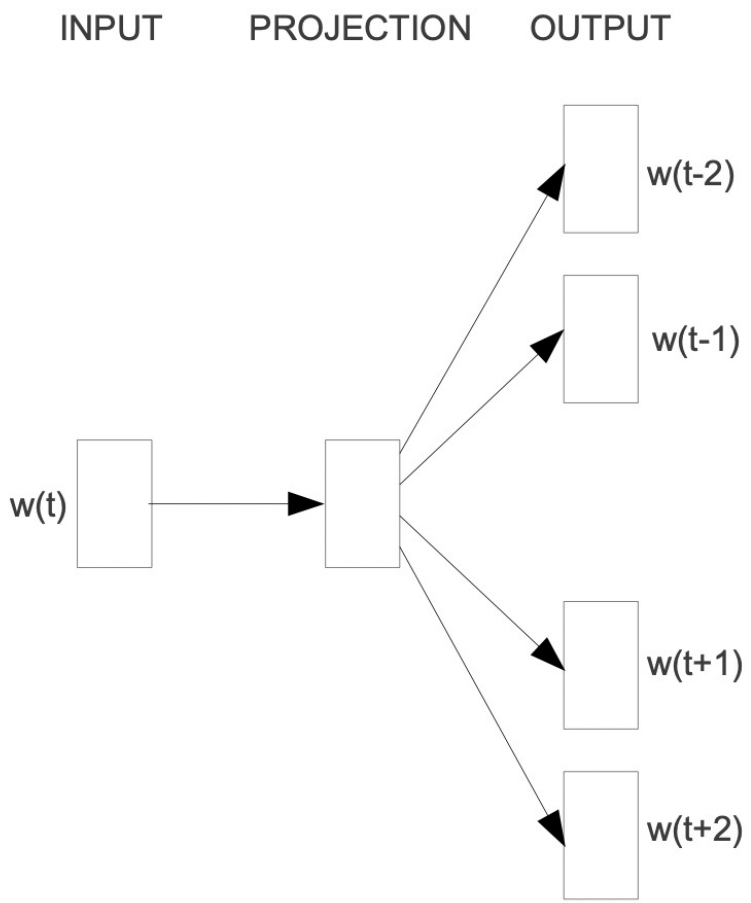
Negative Sampling

- Sampling methods
  - Random sampling$w_j \in \{w_O\} \cup W_{neg}$
  - Distribution sampling: w_j is sampled from $P(w)$
- What is a good P(w)? ➔ less frequent words sampled more often
- Empirical setting: unigram model raised to the power of 3/4

Word	Probability to be sampled for “neg”
is	$0.9^{3/4} = 0.92$
constitution	$0.09^{3/4} = 0.16$
bombastic	$0.01^{3/4} = 0.032$



CBOW



Skip-gram

Mikolov et al. (2013) "Efficient estimation of word representations in vector space."

Comparison

- Count-based
  - LSA, HAL (Lund & Burgess), COALS (Rohde et al), Hellinger-PCA (Lebret & Collobert)
  - Pros
    - Fast training
    - Efficient usage of statistics
  - Cons
    - Primarily used to capture word similarity
    - Disproportionate importance given to large counts
- Direct prediction
  - NNLM, HLBL, RNN, Skipgram/CBOW (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)
  - Pros
    - Generate improved performance on other tasks
    - Capture complex patterns beyond word similarity
  - Cons
    - Benefits mainly from large corpus
    - Inefficient usage of statistics

Combining the benefits from both worlds ➔ GloVe

GloVe

- Idea: ratio of co-occurrence probability can encode meaning
- P_{ij} is the probability that word w_j appears in the context of word w_i

$$P_{ij} = P(w_j \mid w_i) = X_{ij}/X_i$$

- Relationship between the words w_i and w_j

	x=solid	x=gas	x=water	x=random
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{stream})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{stream})}$	8.9	8.5×10^{-2}	1.36	0.96

Pennington et al. (2014) “Glove: Global vectors for word representation.”

GloVe

- The relationship of w_i and w_j approximates the ratio of their co-occurrence probabilities with various w_k

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F((v_{w_i} - v_{w_j})^T v'_{\tilde{w}_k}) = \frac{P_{ik}}{P_{jk}} \qquad F(\cdot) = \exp(\cdot)$$

$$v_{w_i} \cdot v'_{\tilde{w}_k} = v_{w_i}^T v'_{\tilde{w}_k} = \log P(w_k \mid w_i)$$

$$v_{w_i} \cdot v'_{\tilde{w}_j} = v_{w_i}^T v'_{\tilde{w}_j} = \log P(w_j \mid w_i) \qquad P_{ij} = X_{ij}/X_i$$

$$= \log P_{ij} = \log(X_{ij}) - \log(X_i)$$

$$v_{w_i}^T v'_{\tilde{w}_j} + b_i + \tilde{b}_j = \log(X_{ij})$$

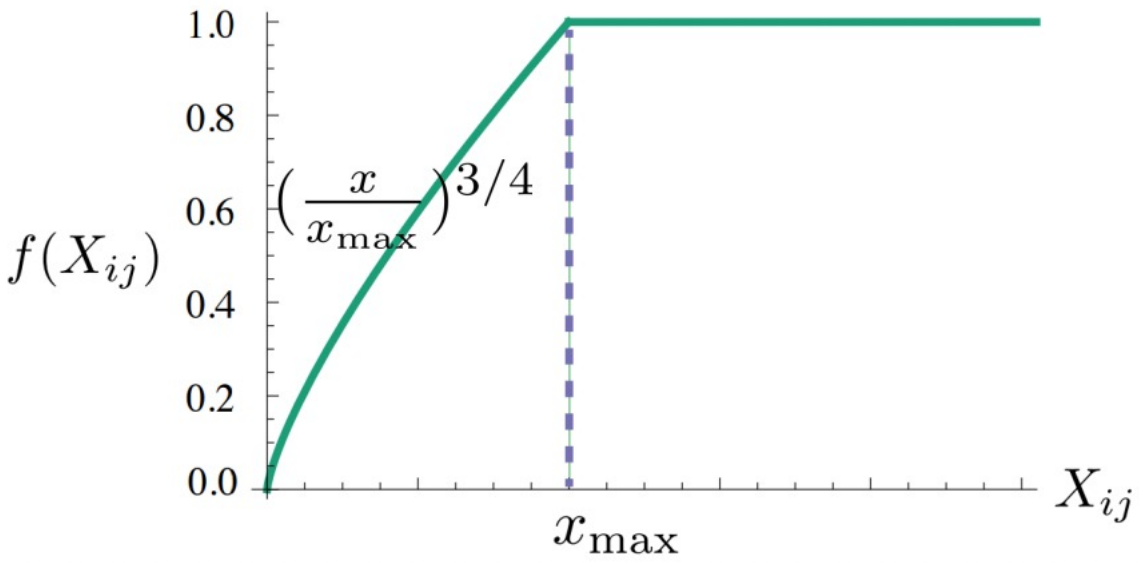
$$C(\theta) = \sum_{i,j=1}^{V} f(P_{ij})(v_{w_i} \cdot v'_{\tilde{w}_j} - \log P_{ij})^2$$

$$C(\theta) = \sum_{i,j=1}^{V} f(X_{ij})(v_{w_i}^T v'_{\tilde{w}_j} + b_i + \tilde{b}_j - \log X_{ij})^2$$

GloVe – Weighted Least Squares Regression Model

$$C(\theta) = \sum_{i,j=1}^{V} f(X_{ij})(v_{w_i}^T v'_{\tilde{w}_j} + b_i + \tilde{b}_j - \log X_{ij})^2$$

- Weighting function should obey
  - $f(0) = 0$
  - $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted
  - $f(x)$ should be relatively small for large values of x, so that frequent co-occurrences are not overweighted



Computing word similarity: Dot product and cosine

- The dot product between two vectors is a scalar:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- The dot product tends to be high when the two vectors have large values in the same dimensions
- Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

- Dot product favors long vectors
- Dot product is higher if a vector is longer (has higher values in many dimension)
- Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

- Frequent words (of, the, you) have long vectors (since they occur many times with other words).
- So dot product overly favors frequent words

Cosine for computing word similarity

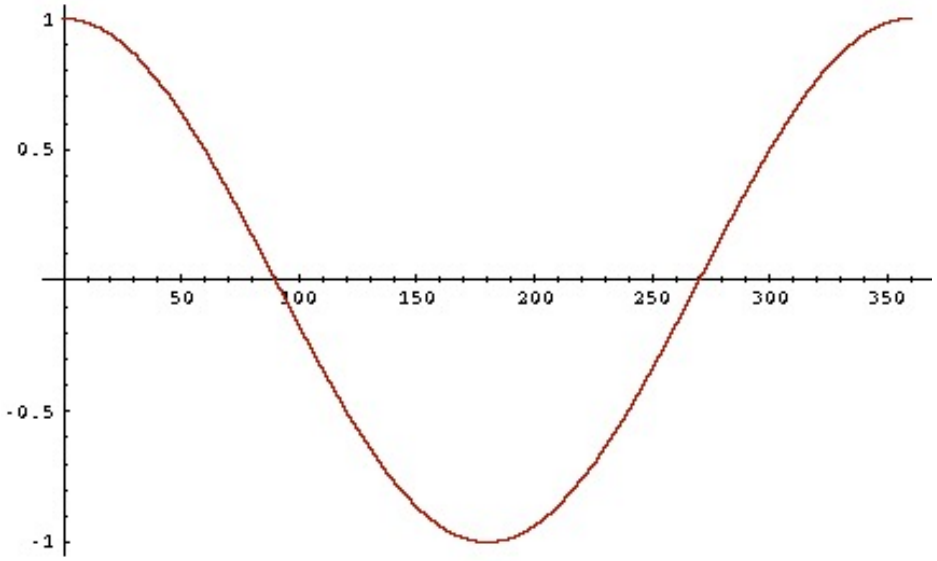
$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Based on the definition of the dot product between two vectors a and b

$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$
$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

Cosine as a similarity metric

- -1: vectors point in opposite direction
- +1: vectors point in same directions
- 0: vectors are orthogonal



- But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

The kinds of neighbors depend on window size

- **Small windows** (C= +/- 2) : nearest words are syntactically similar words in same taxonomy

  Hogwarts nearest neighbors are other fictional schools

  Sunnydale, Evernight, Blandings

- **Large windows** (C= +/- 5) : nearest words are related words in same semantic field

  Hogwarts nearest neighbors are Harry Potter world:

  Dumbledore, half-blood, Malfoy

Analogical relations

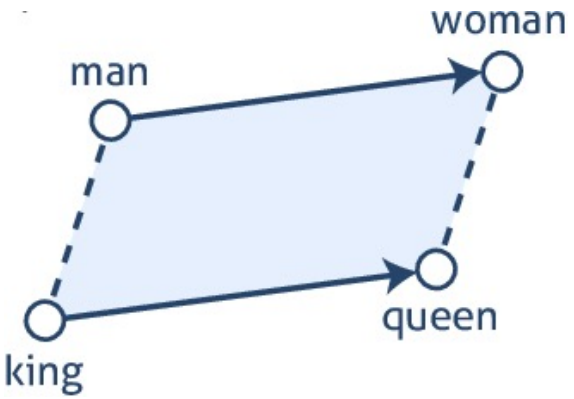
- For a problem a:a* = b:b*, the parallelogram method is:

$$\hat{b}^* = \underset{x}{\text{argmax}}\ \text{distance}(x, a^* - a + b)$$

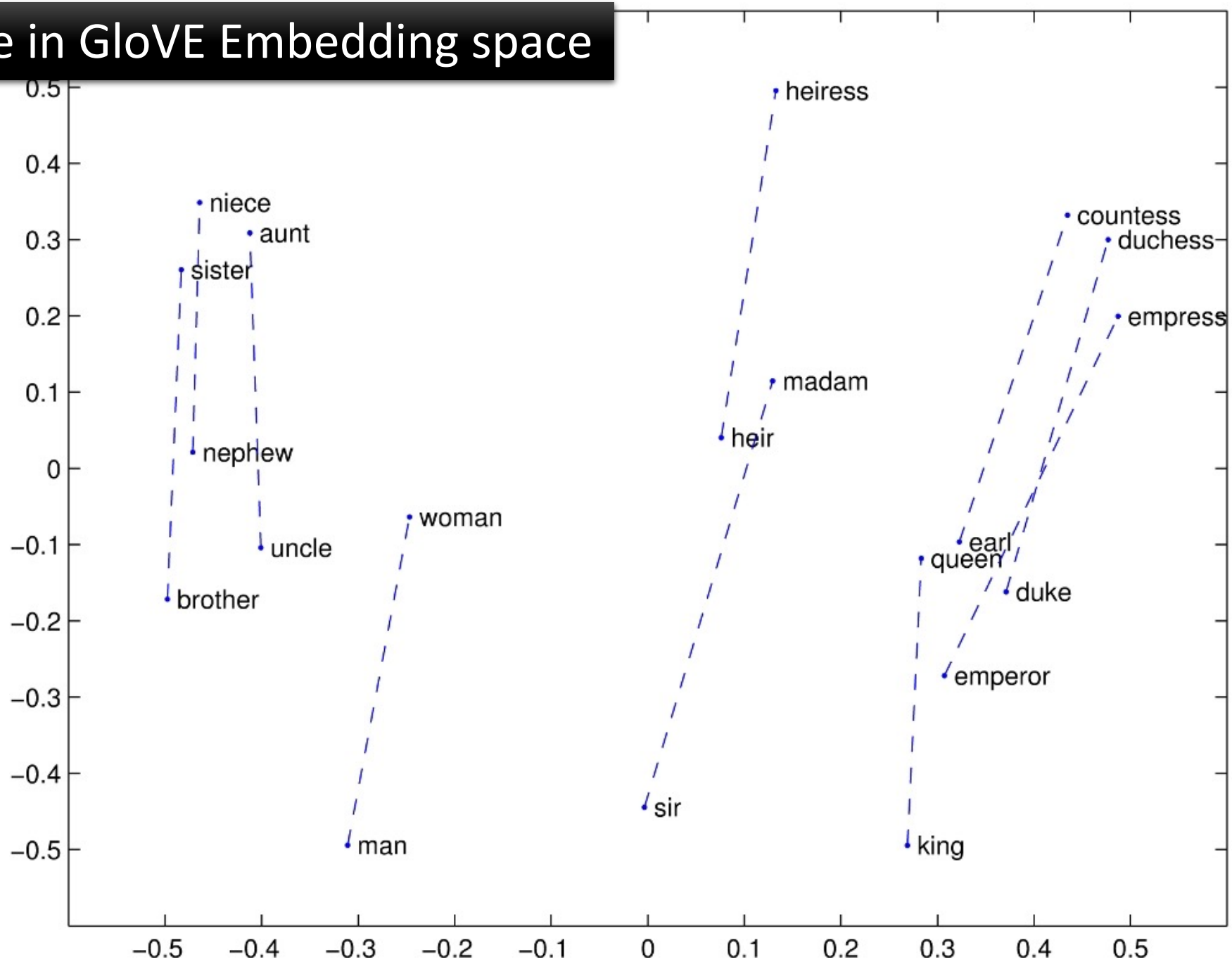
- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$

$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ is close to $\overrightarrow{\text{Rome}}$



Structure in GloVE Embedding space



Embeddings reflect cultural bias

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

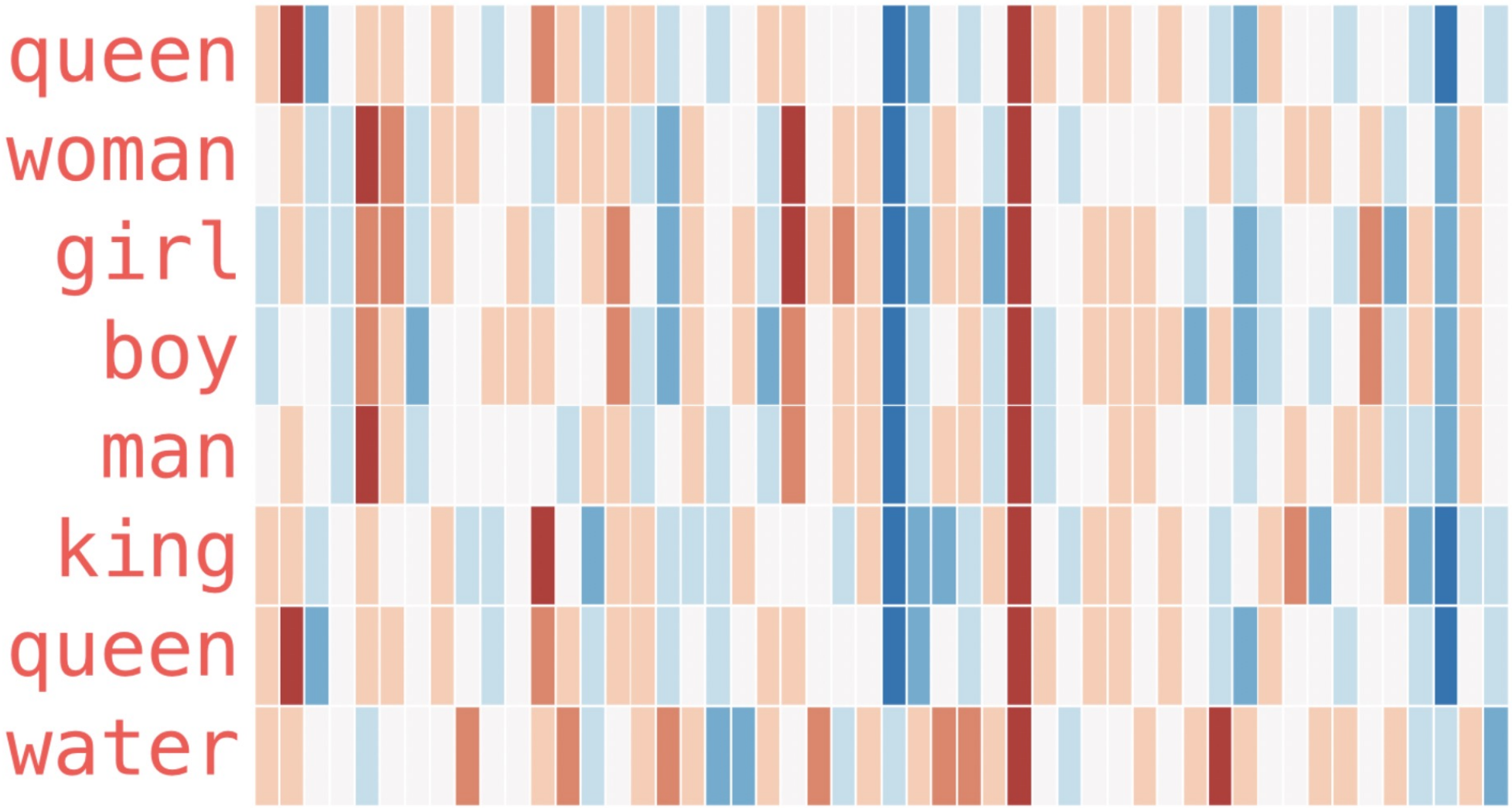
Resources of Word Embeddings

- Word2vec (Mikolov et al., 2013)
  - https://code.google.com/archive/p/word2vec/
- GloVe (Pennington et al., 2014)
  - http://nlp.stanford.edu/projects/glove/
- One of the well-known packages—Gensim 👍
  - https://radimrehurek.com/gensim/apiref.html

Reference

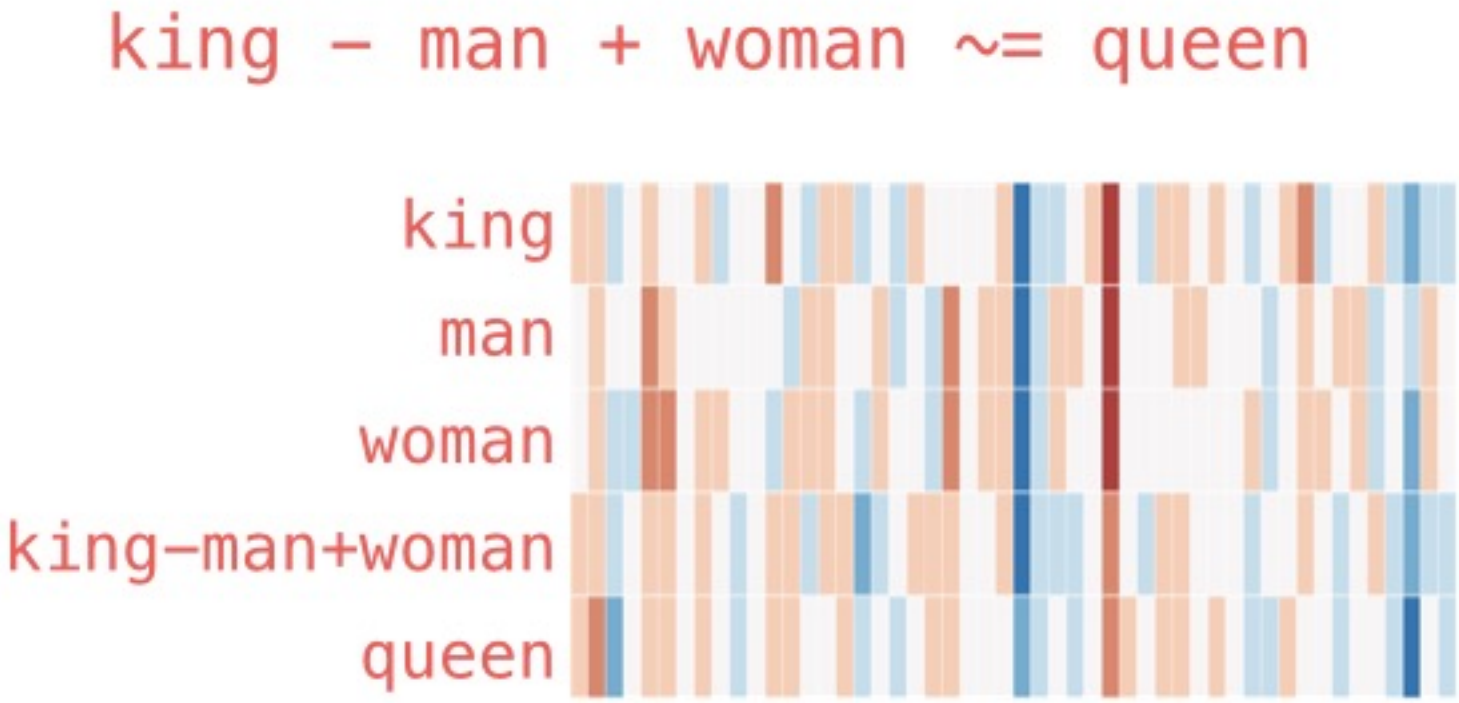
- Lecture Slides from Applied Deep Learning by Prof. Yun-Nung Chen
  - https://www.csie.ntu.edu.tw/~miulab/s110-adl/doc/220307_WordEmbeddings.pdf
- Lecture Slides from Machine Learning by Prof. Hung-yi Lee
  - https://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/word2vec%20(v2).pdf
- Lecture Slides from Information Retrieval and Extraction by Prof. Hsin-Hsi Chen
- Natural Language Processing with Classification and Vector Spaces
  - https://www.coursera.org/learn/classification-vector-spaces-in-nlp
- Lecture Slides from the Stanford Coursera course by Prof. Dan Jurafsky and Prof. Christopher Manning
  - https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html
- McCormick, C. (2016, April 19). Word2Vec Tutorial - The Skip-Gram Model. Retrieved from http://www.mccormickml.com

Appendix



Source of image: https://jalammar.github.io/illustrated-word2vec/

- The resulting vector from "king-man+woman" doesn't exactly equal "queen", but "queen" is the closest word to it from the 400,000 word embeddings we have in this collection.



Source of image: https://jalammar.github.io/illustrated-word2vec/